

MIYAMOTO et al.  
Appl. No. Unknown  
January 15, 2004

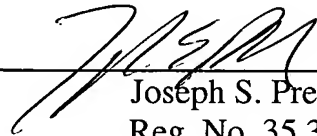
**REMARKS/ARGUMENTS**

Favorable examination is earnestly solicited.

Respectfully submitted,

**NIXON & VANDERHYE P.C.**

By: \_\_\_\_\_



Joseph S. Presta  
Reg. No. 35,329

JSP:mg  
1100 North Glebe Road, 8th Floor  
Arlington, VA 22201-4714  
Telephone: (703) 816-4000  
Facsimile: (703) 816-4100

## **TITLE OF THE INVENTION**

Video Game Apparatus and Information Storage Medium for Video Game

## **BACKGROUND OF THE INVENTION**

### Field of the invention

This invention relates to a video game apparatus and a game program memory medium therefor. In particular, the invention relates to a video game apparatus which generates, and supplies to a display, an image signal to display a player object existing on a land object in a virtual three dimensional space by virtue of player object data and land object data, and a game program memory medium to be used therefor.

### Description of the prior art

In a conventional video game machine, when a player wishes a player object to jump, the player presses a jump button on a controller so that the CPU causes the player object to jump in response to jump button operation. That is, when the player object is caused to jump over an obstacle, such as a hollow or hole, the player must press the jump button in front of the hollow or hole while manipulating a move direction instructing means, such as a joystick or cross button. However, it may be the case that the player object is unsuccessful in jumping across the obstacle, as the timing may be off in pressing the jump button, or in the positioning of the player object. Skillful operation of a jump

button has been required to make the player object jump up and get across an obstacle.

Meanwhile, complicated button operation is necessary to cause the player object to perform other actions than jump (e.g., opening and closing a door or climbing stairs, etc.). The player may experience difficulty in the game progression because his or her attention is stuck to button manipulation.

Such games, called action games, are becoming more difficult to play every year. They are too difficult for the player. In particular, there is a tendency for beginners to avoid action games.

## **SUMMARY OF THE INVENTION**

Therefore, it is a primary object of the present invention to provide a novel video game apparatus, and a program memory medium to be used therefor.

Additionally, it is an object of the present invention to provide a novel video game apparatus which is easy for a player to operate a player object, and a game program memory medium to be used therefor.

Also, it is an object of the present invention to provide a video game apparatus with which a player object can get over an obstacle(s) without difficulty, and a game program memory medium to be used therefor.

It is another object of the present invention to provide a video game apparatus wherein it is possible for a player object, virtual camera, or the like, to

automatically carry out a required operation such as jumping, camera switching, or the like, and a game program memory medium to be used therefor.

Additionally, it is an object of the present invention to provide a video game apparatus which can effect complicated control with a simple program, and a game program memory medium to be used therefor.

A video game apparatus, according to the present invention, generates and supplies to a display, an image signal for displaying a player object existing on a land object in a virtual three dimensional space by processing image data for the player object and the land object according to a program. The video game apparatus comprises: a player object image data generating means for generating player object image data to display a player object and a land object image data generating means for generating land object image data to display a land object, wherein the land object image data includes a program control code. The video game apparatus further comprises: a program control code detecting means to detect the program control code in relation to position of the player object and an image changing means to cause the image signal to change depending upon the program control code detected.

The program control code includes an action code to control an action of the player object. The image changing means includes animation data output means to output animation data that automatically causes the player object to make an action in accordance with the action code.

Specifically, if the land object is a hollow or hole and the action code is "jump," the animation data output means outputs animation data that causes the player object to jump over the hollow or hole.

In one embodiment, the video game apparatus has a controller including a direction instruction means to direct the movement of the player object. The video game apparatus further comprises: a moving speed detecting means for detecting the moving speed of the player object, a jump distance calculating means for calculating the jump distance of the player object based on the moving speed, and the animation data output means for outputting animation data causing the player object to make jumping action according to the jump distance.

If the land object is a wall surface and the action code is "climb," the animation data output means outputs animation data such that the player object makes an action of climbing the wall surface.

However, if the action code is not "climb," a wall surface height calculating means calculates a height of the wall surface, and the animation data output means outputs animation data so that the player object makes an optimal action in accordance with the height of the wall surface.

In an embodiment of the present invention, the program control code includes a camera control code, and image changing means, including a camera control means to control a virtual camera provided in the three dimensional virtual space.

The virtual camera is selected from a plurality of virtual cameras. The camera control code includes a camera switching code and camera control means, including a camera switching means to switch between virtual cameras depending upon the camera switching code.

The program control code includes a sound code, and the video game apparatus further comprises a sound data generating means to generate sound data and a sound control means to control sound to be outputted from the sound data generating means depending upon the sound code.

The sound data generating means can generate sound data for a plurality of sounds. The sound code includes a sound switching code and sound control means, including a sound switching means to switch the sound data depending upon the sound switching code.

It is possible to control only sound with a program control code. In this case, a video game apparatus generates and supplies to a display, an image signal to display a player object existing on a land object in a virtual three dimensional space by processing image data for the player object and land object according to a program, and the apparatus further supplies a sound signal to a sound output means by processing sound data according to a program. The video game apparatus comprises: a player object image data generating means for generating player object image data to display a player object and a land object image data generating means for generating land object image data to display a land object. The land object image data includes a program control

code. The video game apparatus further comprises: a program control code detecting means to detect the program control code with respect to the position of the player object and a sound changing means to cause the sound signal to change according to the program control code detected.

Generally, the video game apparatus uses a memory medium to store a game program or image data. A memory medium according to the present invention is applicable to a video game apparatus for generating, and supplying to a display, an image signal to display a player object existing on a land object in a virtual three dimensional space by processing image data for the player object and the land object according to a program. The video game apparatus further stores a program to be processed by an information processing means in the same apparatus. The memory medium comprises: a player object image data generating program to generate player object image data for displaying a player object and a land object image data generating program for generating land object image data to display a land object. The land object image data includes a program control code. The memory medium further comprises: a program control code detecting program for detecting the program control code with respect to the position of the player object and an image changing program for causing the image signal to change according to the program control code detected.

The game program memory medium is formed with an image data area. The image data area stores player object data and land object data. The player

object data includes polygon data representing shape and animation data corresponding to an action state. The land object data includes polygon data representing a shape and attribute data. This attribute data contains a program control code, which includes an action code, a camera code and a sound code. The game memory medium further includes a program to process image data. The video game apparatus generates a game according to the image data and programs, in conjunction with controller data from the controller. Accordingly, a display screen shows a game image of a player object existing on a land object in a virtual three dimensional space.

If the player object approaches or exists on a relevant land object, a detecting menu detects a program control code contained in the land object image data. Accordingly, the image changing means, which is different from a usual program, controls an action of the player object or a virtual camera.

If an action code is contained in the land object data that represents a land object in the vicinity of the player object, an action code detecting means (i.e., an action code detecting program) detects an action code. The animation data output means (i.e., an animation data output program) outputs animation data causing the player object to make an action in accordance with the detected action code. Therefore, it is possible for the player object to automatically perform an optimal action corresponding to the detected action code.



Specifically, if the land object is a hollow or hole and the action code is "jump," the animation output means (animation data output program) outputs animation data causing the player object to jump across the hollow or hole.

If the player object moves according to a direction instructing means on the controller, the moving speed detecting means (i.e., moving speed detecting program) detects the moving speed of the player object, and the jump distance detecting means (i.e., jump distance detecting program) calculates a jump distance of the player object. In this case, the animation data output means (i.e., animation data output program) outputs animation data causing the player object to jump depending upon the jump distance.

If the land object is a wall surface and the action code is "climb," the animation data output means (i.e., animation data output program) outputs animation data causing the player object to climb the wall surface.

If a wall surface height is calculated by a wall surface height calculating means (i.e., wall height detecting program), it is determined under which range of  $0 < H \leq 25$ ,  $25 < H \leq 50$ ,  $50 < H \leq 100$  or  $100 < H \leq 150$  the wall surface height (H) falls. The animation data output means (i.e., animation data output program) outputs animation data causing an optimal action for the wall surface height.

Furthermore, if the program control code is a camera code, a plurality of virtual cameras properly arranged in the virtual three dimensional space are selectively activated by the camera code (i.e., camera switching code).

Also, if the program control code is a sound code, a sound data generating means switches the sound data to be produced.

According to the present invention, required control can be automatically made in accordance with a control code contained in the land object image data, including, player object action control, image change control including virtual camera switching, and sound control such as sound data switching. Even in the case that a player object or a camera is controlled in a complicated manner according to the position of the player object, it is easy to design a program.

For example, if the program control code is an action code, it is very easy for a player to manipulate a player object. If the action code is "jump," the player object automatically jumps. Thus, the player object can easily get across an obstacle such as a hole or hollow. If the action code is "climb," the player object can automatically climb a wall surface. If the action code is "door," a door automatically opens and the player object is allowed to enter the door. Further, if the action code is "ladder," the player object will automatically climb a ladder.

The above described objects and other objects, features, aspects and advantages of the present invention will become more apparent from the following detailed description of the present invention when taken in conjunction with the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic illustrative view showing a video game system of one embodiment of this invention;

Figure 2 is a block diagram showing in detail a video game machine of the Figure 1 system;

Figure 3 is a block diagram showing in detail a controller control circuit of the Figure 2 video game machine;

Figure 4 is a block diagram showing in detail a controller and controller pack for the Figure 2 video game machine;

Figure 5 is an illustrative view showing a memory map of an external ROM for the Figure 2 video game machine;

Figure 6 is an illustrative view showing a memory map of RAM for the Figure 2 video game machine;

Figure 7 is a flowchart showing an overall operation of the Figure 1 embodiment;

Figure 8 is a flowchart showing in detail a land object process in the Figure 7 flowchart;

Figure 9 is a flowchart showing in detail one part of an action determining process in the Figure 7 flowchart;

Figure 10 is a flowchart showing in detail an action determining process for the case of a hole in the Figure 9 flowchart;

Figure 11 is an illustrative view showing one example of a jump (big jump) action to be achieved in the Figure 10 flowchart;

Figure 12 is an illustrative view showing one example of a jump (middle jump) action to be achieved in the Figure 10 flowchart;

Figure 13 is an illustrative view showing one example of a jump (small jump) action to be achieved in the Figure 10 flowchart;

Figure 14 is an illustrative view showing one example of a "not-fall" action in the Figure 10 flowchart;

Figure 15 is a flowchart showing in detail an action determining process for the case of a wall surface in the Figure 9 flowchart;

Figure 16 is an illustrative view showing one example of a wall climbing action to be achieved by the Figure 15 flowchart;

Figure 17 is a flowchart showing one example of a step mounting action to be achieved by the Figure 15 flowchart;

Figure 18 is an illustrative view showing a jump up action to be achieved by the Figure 15 flowchart;

Figure 19 is an illustrative view showing one example of a light climb action to be achieved by the Figure 15 flowchart;

Figure 20 is an illustrative view showing one example of a usual climb action to be achieved by the Figure 15 flowchart;

Figure 21 is an illustrative view showing one example of a hard climb action to be achieved by the Figure 15 flowchart;

Figure 22 is a flowchart showing in detail a door action in the Figure 9 flowchart;

Figure 23 is an illustrative view showing one example of a door action to be achieved by the Figure 22 flowchart;

Figure 24 is a flowchart showing in detail a ladder action in the Figure 9 flowchart;

Figure 25 is an illustrative view showing one example of a ladder action achieved by the Figure 24 flowchart;

Figure 26 is a flowchart showing in detail a player object process in the Figure 7 flowchart;

Figure 27 is an illustrative view showing in detail a camera determining process in the Figure 7 flowchart;

Figure 28 is an illustrative view showing one example of camera arrangement as an example for the camera determining process of Figure 27 flowchart;

Figure 29 is a flowchart showing in detail a first camera control program in the Figure 27 flowchart;

Figure 30 is an illustrative view showing a player object taken by a first camera according to the Figure 29 flowchart;

Figure 31 is a flowchart showing in detail a second camera, or fifth camera control program in the Figure 27 flowchart;

Figure 32 is a flowchart showing in detail a third camera control program of the Figure 27 flowchart;

Figure 33 is an illustrative view showing a player object taken by the third camera according to the Figure 32 flowchart;

Figure 34 is a flowchart showing in detail a fourth camera control program in the Figure 27 flowchart;

Figure 35 is an illustrative view showing a player object taken by a fourth camera according to the Figure 34 flowchart;

Figure 36 is an illustrative view showing a player object taken by the fourth camera according to the Figure 34 flowchart; and

Figure 37 is a flowchart showing in detail a sound process in the Figure 7 flowchart.

## **DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS**

Referring to Figure 1, a video game apparatus in this embodiment includes video game machine 10, ROM cartridge 20 as one example of an information memory medium, display unit 30 connected to video game machine 10, and controller 40. Controller 40 is dismountably connected to a controller pack 50.

Controller 40 includes a plurality of switches or buttons provided on housing 41 in a form graspable by both or one hand. Specifically, controller 40 includes handles 41L, 41C, 41R downwardly extending respectively from a left

end, a right end and a center of housing 41, providing an operation area on a top surface of housing 41. In the operation area, there are provided analog-inputtable joystick (hereinafter referred to as "analog joystick") 45 at a central lower portion thereof, cross-shaped digital direction switch (hereinafter called "cross switch") 46 on the left side, and button switches 47A, 47B, 47C, 47D, 47E and 47F on the right side.

Analog joystick 45 is used to input a moving direction and/or moving speed or moving amount of the player object (object to be operated by a player through controller 40) as determined by an amount and direction of joystick inclination. Cross switch 46 is used to designate a moving direction of the player object, in place of the joystick 45. Button switches 47A and 47B are used to designate a motion of the player object. Button switches 47C - 47D are used to switch over a visual point of a three-dimension image camera or adjust speed or the like of the player object.

Start switch 47S is provided near the center of the operation area. Start switch 47S is operated when starting a game. Switch 47Z is provided at a backside of central handle 41C. Switch 47Z can be utilized as a trigger switch in a shooting game. Switches 47L and 47R are provided at upper left and right of a lateral surface of housing 41.

Button switches 47C - 47F can also be used to control the motion and/or moving speed (e.g. acceleration or deceleration) of the player object in a shoot or action game, in addition to switching the camera visual point. Moreover,

switches 47A - 47F, 47S, 47Z, 47L and 47R can be arbitrarily defined in their function depending upon a game program.

Figure 2 is a block diagram of the video game system of the Figure 1 embodiment. Video game machine 10 contains central processing unit (hereinafter referred to as "CPU") 11 and coprocessor (reality coprocessor; hereinafter referred to as "RCP") 12. RCP 12 includes bus control circuit 121 for controlling buses, signal processor (reality signal processor; hereinafter referred to as "RSP") 122 for performing polygon coordinate transformation, shading treatment and so on, and rendering processor (reality display processor; hereinafter referred to as "RDP") 46 for rasterizing polygon data into an image to be displayed and converting the same into a data form (dot data) memorable on a frame memory.

RCP 12 is connected to ROM cartridge 20 with external ROM 21 by cartridge connector 13, a disc-drive connector 197 (not shown) for detachably mounting a disc drive 29 (not shown), and RAM 14. Also, RCP 12 is connected with DAC (Digital/Analog Converters) 15 and 16 for respectively outputting a sound signal and video signal to be processed by CPU 11. Further, RCP 12 is connected with controller control circuit 17 to serially transfer operating data on one or more of controllers 40 and/or controller pack 50.

Bus control circuit 121 included in RCP 12 performs parallel/serial conversion on a command supplied in a parallel signal from CPU 11 via a bus, thereby supplying a serial signal to the controller control circuit 17. Also, bus



control circuit 121 converts a serial signal inputted from controller control circuit 17 into a parallel signal, giving an output to CPU 11 via the bus. The data representative of an operating state (operating signal or operating data) read out of controller 40A - 40D is processed by CPU 11, and temporarily stored within RAM 14, and so on. In other words, RAM 14 includes a storage site for temporarily memorizing the data to be processed by CPU 11, so that it is utilized for smoothly reading and writing data through bus control circuit 121.

Sound DAC 15 is connected to speaker 31 by connector 19a provided at a rear face of the video game machine 10. Video DAC 16 is connected to display 30, such as a TV receiver or CRT, by connector 19b provided at the rear face of video game machine 10.

Controller control circuit 17 is connected to controller 40 by connector 18, which is on the front face of video game machine 10. The connector 18 is disconnectably connected by a controller 40 through a connecting jack. This electrical connection enables transmission/reception or transfer of data between controller 40 and video game machine 10.

Controller control circuit 17 is used to transmit and receive data in serial between RCP 12 and connector 18. Controller control circuit 17 includes, as shown in Figure 3, data transfer control circuit 171, transmitting circuit 172, receiving circuit 173 and RAM 174 for temporarily memorizing transmission and reception data. Data transfer control circuit 171 includes a parallel/serial converting circuit and a serial/parallel converting circuit in order to convert a

data format during data transfer. Also, data transfer control circuit 171 performs write/read control on RAM 174. The serial/parallel converting circuit converts the serial data supplied from RCP 12 into parallel data, supplying it to RAM 174 or transmitting circuit 172. The parallel/serial converting circuit converts the parallel data supplied from RAM 174 or receiving circuit 173 into serial data, to supply it to RCP 12. Transmitting circuit 172 converts the command for reading signals from controller 40 and the writing data (parallel data) to controller pack 50, into serial data to be delivered to channels CH1 - CH4 corresponding to respective controllers 40. Receiving circuit 173 receives, in serial data, operational state data of the controllers inputted through corresponding channels CH1 - CH4 and data read from controller pack 50, and converts the serial data into parallel data to be delivered to data transfer control circuit 171. The data transfer control circuit 171 writes into RAM 174 data transferred from RCP 12, data of the controller from the receiving circuit 173, or data read out of RAM controller pack 50. Additionally, the data transfer control circuit 17 reads data out of the RAM 174 based on a command from RCP 12 so as to transfer it to RCP 12.

RAM 174 includes memory sites for the respective channels CH1 - CH4 (not shown). Each of the memory sites is stored with a command for the channel, transmitting data and/or reception data.

Figure 4 is a detailed circuit diagram of controller 40 and controller pack 50. The housing of controller 40 incorporates operating signal processing

circuit 44, etc. in order to detect an operating state of joystick 45, switches 46, 47, etc. and transfer the detected data to controller control circuit 17. Operating signal processing circuit 44 includes receiving circuit 441, control circuit 442, switch signal detecting circuit 443, counter circuit 444, joyport control circuit 446, reset circuit 447 and NOR gate 448. Receiving circuit 441 converts a serial signal, such as a control signal transmitted from controller control circuit 17 or writing data to controller pack 50, into a parallel signal to supply it to control circuit 442. When the control signal transmitted from controller control circuit 17 is a signal for resetting the X and Y coordinates of the joystick 45. The control circuit 442 sends a reset signal through NOR gate 448, to reset count values of X-axis counter 444X and Y-axis counter 444Y within counter 444, to 0.

Joystick 45 includes X-axis and Y-axis photo-interrupters in order to include a lever inclination into X-axis and Y-axis components, generating pulses in number proportional to the inclination. The pulse signals are respectively supplied to counter 444X and counter 444Y. Counter 444X counts a number of pulses generated in response to an inclination amount when joystick 45 is inclined in the X-axis direction. Counter 444Y counts a number of pulses generated responsive to an inclination amount when joystick 45 is inclined in the Y-axis direction. Accordingly, the resultant X-axis and Y-axis vectors determined by the count values of counters 444X and 444Y serve to determine a moving direction and a coordinate position of the player object

which may represent a hero character or a cursor. Counters 444X and 444Y are reset, when a reset signal is supplied from reset signal generating circuit 447 upon turning on the power or a reset signal is supplied from switch signal detecting circuit 443 by simultaneous depression of predetermined two switches.

Switch signal detecting circuit 443 responds to a switch-state output command supplied at an interval of a constant period (e.g., the same 1/30 second interval as a TV frame period) from control circuit 442, to read a signal varying depending upon a depression state of cross switch 46 and switches 47A - 47Z (not shown). The read signal is delivered to control circuit 442. Control circuit 442 responds to a read-out command signal of operational state data from controller control circuit 17 to supply, in a predetermined data format, the operational state data on switches 47A - 47Z and count values of counters 444X and 444Y to transmitting circuit 445. Transmitting circuit 445 converts the parallel signal outputted from control circuit 442 into a serial signal, and transfers it to controller control circuit 17 via converting circuit 43 and signal line 42. Control circuit 442 is connected with joystick control circuit 446 via an address bus and a data bus as well as port connector 46. Joystick control circuit 446 performs data input/output (or transmission/reception) control according to a command from CPU 11 when controller pack 50 is connected to port connector 46.

Controller pack 50 is structured by connecting RAM 51 to the address bus and data bus. RAM 51 is also connected to battery 52. The RAM 51 stores backup data in relation to a game and backup data is stored using electric power from battery 52 if controller pack 50 is withdrawn from port connector 46.

Figure 5 is a memory map illustrating a memory space of external ROM 21 incorporated in ROM cartridge 20 in Figures 1 and 2. External ROM 21 includes a plurality of memory areas ("areas"), such as program area 22, image data area 23 and sound memory area 24. The areas are previously stored with various programs.

Program area 22 stores a program required to process game images, game data suited for a game content, and so forth. Specifically, program area 22 includes memory areas 22a - 22i to a CPU 11 operation program. Main program area 22a stores a main routine processing program for a game shown in Figure 7. Controller data determining program area 22b stores a program to process controller 40 operation data. Land object program area 22c stores a program to display and control a land object in the vicinity of the player object. Player object program area 22d stores a program to display and control an object by a player ("player object").

Program area 22 further includes control code detecting program area 22e. A program is installed on area 22e to detect a control code contained in land object image data. Camera control program area 22f stores a camera control program that controls which direction and/or position a moving object,

including the player object or background object, takes in a three dimensional space. In the embodiment, a plurality of virtual cameras are installed in a three dimensional space. Accordingly, camera control program area 22f includes a first camera control program, second camera control program, ..., Nth camera control program to individually control one to N virtual cameras, respectively.

Action control program area 22g stores a program to read out animation data contained in the player object image data, which causes the player object to act according to a control code detected by a control code detecting program. The action control program includes various calculation programs. The calculation programs include a moving speed detecting program that detects the moving speed of the player object, a jump distance calculating program that calculates the jump distance of the player object based on a moving speed, and a wall height calculating program that calculates a wall height. This action control program determines an action for the player object according to an action code, control code or calculation program, and reads animation data out of image data area 23 depending upon the action. Accordingly, action control program 22g cooperates with image data area 23 to thereby constitute an animation data output program.

Image buffer and Z buffer write program area 22h stores a write program by which CPU 11 causes RCP 12 to write onto an image buffer and a Z buffer. For example, write program area 22h stores a program to write color data to the frame memory area in Figure 6 of the RAM and a program to write depth data

to Z buffer area 204 in Figure 6. These programs write image data based on texture data for plurality of moving objects or background objects to be displayed on one background scene.

Also, sound process program area 22i stores a program to generate a message through sound effects, melodies or voices.

Image data area 23 includes, as shown in Figure 5, two memory areas: 23a and 23b. Memory area 23a stores image data, such as coordinate data and animation data of multiple polygons, on an object-by-object basis, in order to display, with a display control program, a player object and in a predetermined fixed position or to display a movable object. Memory area 23b stores image data, such as various polygon data and attribute data, on an object-by-object basis in order to display, with a display control program, a land object. The attribute data includes an action code representation of an action to be performed by the player object (e.g., jump, wall scramble, door open and close, ladder climb, etc), a kind code representation of a kind of a land polygon (e.g., hole, ice, sand, lava, etc), a melody code representation of a kind of BGM, an enemy code representation (whether or not an enemy is present) and an enemy kind, and a camera code to instruct switching between cameras. These codes are collectively referred to as "control codes". The control codes have been previously set within the polygon data of every polygon representing the land objects. The required land objects include a land object on which the player

object exists, a land object in the vicinity of which the player object exists, and so forth.

Sound memory area 24 stores sound data, such as phrases, sound effects and game melodies, for each scene to output a message as above in a manner suited for a relevant scene. Specifically, BGM1 and BGM2 are stored as game melodies and sound data such as "outcry" as a sound effect.

The memory medium or external memory may use an arbitrary memory medium, such as a CD-ROM or magnetic disc, in place of, or in addition to, ROM cartridge 20. In such a case, a disc drive (not shown) could read, or write as required, various data for a game (including program data and image display data) from the optical or magnetic disc-formed memory medium, such as a CD-ROM or magnetic disc. This disc drive reads out data stored on the magnetic disc or optical disc, and transfers the data to RAM 14.

In this manner, program area 22 is installed with programs so that a game image signal can be created by processing the image data set on image data area 23 in a manner similar to a conventional video game apparatus and a sound signal can be produced by processing the sound data installed on sound memory area 24. In this embodiment, a program control code is previously set on the image data stored in image data area 23 (such as, data in the land object image data). When the detected program control code depends on the player object's position, the animation for the player object changes, the virtual camera switches, and further the sound signal changes to comply with a detected



program control code. Thus, the program control code serves as a program control factor or program change factor.

If a program code is detected, the player object's animation changes or the camera switches. Thus, it is possible to provide image change in a manner different from that by the execution of a usual program. Also, if a program control code is detected, so that the sound signal switches, it is possible to cause a different sound change than by executing an ordinary program.

The control code is explained in more detail. As mentioned above, the land object data includes attribute data, which includes control code. The attribute data is a representation of what the present land object is (e.g., a kind of an object, such as a hole, floor, wall surface, stair, grassy land, etc.). Therefore, CPU 11 can determine a kind of a land object by detecting attribute data.

The control code is configured by 1 or 2 or more bits in attribute data. The attribute data is included within each polygon that represents a land object. As a result, the control data is included in each polygon. The control code represents, by 1 or 2 or more bits, a control content (e.g., "jump", "climb", "enter door", "ladder", "camera switch", "sound switch", etc).

In the above explanation, a kind of a land object was determined by attribute data. However, the method for detecting a land object may be as follows. A land object on which the player object is moving may be detected as a floor object, whereas a land object provided at 90 degrees (vertically) with

respect to the floor object is detected as a wall or wall surface object. In this case, a land object existing above the player object will be detected as a ceiling object. Thus, a kind of a land object may be determined by a positional relationship or angle relative to the player object.

In either case, a program control code (e.g., a control code, action code, camera code, sound code, etc.) is set in attribute data.

Figure 6 is a memory map illustrating an entire memory space of RAM 14. RAM 14 includes memory areas 201 — 209. For example, RAM 14 includes display list area 201, program area 202, frame memory (or image buffer memory) area 203 for temporarily storing one frame of image data, Z buffer area 204 for storing, dot by dot, depth data of the frame memory area data, image data area 205, sound memory area 206, area 207 for storing controller operation state data, working memory area 208, and register/flag area 209. Memory areas 201 — 209 are memory spaces to be accessed through bus control circuit 121 by CPU 11 or directly by RCP 12, and assigned with an arbitrary capacity (or memory space) by a game used. Meanwhile, image data area 205 and sound memory area 206 temporarily store image data or sound data required to execute a program transferred to program area 202, when that program is a part of data game program for one entire scene (stage) stored in the memory area 22 of ROM 21. In this manner, if the program required for a certain scene or data part are stored in memory areas 202, 205, 206, it is

possible to enhance data processing efficiency and image processing speed compared to directly reading from ROM 21 each time CPU 11 requires.

Specifically, frame memory area 203 has a memory capacity corresponding to the number of picture elements (pixels or dots) of display 30 (Figure 1) times the number of bits of color data per pixel. This enables storage of dot-by-dot color data corresponding to the pixels on display 30. Frame memory area 203 temporarily stores color data dot by dot when displaying a moving object (e.g., a player object, fellow object, enemy object, boss object etc.) or other objects (e.g., a land object, background or stationary object, etc.) that are stored in image data area 105.

Z buffer area 204 has a memory capacity corresponding to the number of picture elements (pixels or dots) of display 30 times the number of bits of depth data per pixel. This enables storage of dot-by-dot depth data corresponding to each pixel on display 30. Z buffer area 204 temporarily stores depth data dot by dot when displaying a moving and/or stationary object (e.g., a moving object such as a player object, fellow object, enemy object, boss object, etc.) and other objects (e.g., a land object, background, or stationary object or the like that are memorized in image data area 205).

Image data area 205 stores coordinate data and texture data for polygons for each of stationary and/or movable objects stored in ROM 21. For example, one course or stage of data is transferred from ROM 21 in advance of their

image processing. Additionally, image data area 205 stores animation data that has been read out from image data area 23 of external ROM 21.

Sound memory area 206 is transferred part of the sound data (data representing a phrase, melody and sound effects) stored in the memory area of ROM 21 and temporarily stores it as sound data is played through sound producing unit 32.

Controller data (operation state data) memory area 207 temporarily stores operation state data that is a representation of an operation state read from controller 40.

Working memory area 208 temporarily stores data, such as parameters during execution of a program by CPU 11.

Register/flag area 209 includes register area 209r and flag area 209f. Register area 209r is formed with multiple registers that are individually loaded with data (not shown). Register area 209r is formed with multiple flags to be separately set or reset (not shown).

Figure 7 is the main flowchart of the video game system in this embodiment. If the power is turned on, in step S1, CPU 11 starts video game machine 10 in a predetermined initial state. For example, CPU 11 transfers a starting program of the game programs stored on program area 22 of external ROM 21 to program area 202 of RAM 14, and sets parameters to their initial values. Then, CPU 11 begins executing the sequential steps of Figure 7.

The operation of the main flowchart of Figure 7 is carried out at a frequency of every frame (1/60th second) or every few frames. The steps S2 - S12 are executed repeatedly until the course has been cleared. If the game ends without successfully clearing the course, a game over process is performed in step S14 (following step S13). If the course is cleared successfully, the process returns to step S1.

That is, a game course screen and/or course selecting screen is displayed in step S1. However, if the game is started after turning on the power, the first course screen is displayed. If the first course is cleared, the next course is displayed.

A controller process is carried out in step S2. In this process, the operation of joystick 45, cross switch 46 and switches 47A - 47Z are detected. The operation state detection data (controller data) is read, then the controller data is written onto controller data area 141 of RAM 14.

In step S3, a land object process is performed and is explained in detail by the subroutine of Figure 8 below. This process includes a calculation of a land object display position and shape based on a program partly transferred from memory area 22c and land object polygon data transferred from the memory area (Figure 5).

In step S4, a process is executed to determine an action for the player object and is explained in detail by Figures 9 to 26 below. Determination is

made on an action for the player object according to a control code or action code that was explained above.

In step S5, a process is performed to display a player object. This process basically causes changes in position, direction, shape and location on the basis of the operating state of joystick 45 (controller data) operated by a player and the presence or absence of enemy attack. For example, the polygon data after change is determined by a calculation based on the program transferred from memory area 22e in Figure 5 of external ROM 21, the player object polygon data transferred from memory area 23a, and the controller data (i.e., joystick 45 operating state). Colors are given by texture data to the polygons calculated by the above procedure.

Step S6 performs a camera determination process. This step determines which virtual camera is chosen to take pictures of an object in a virtual three dimensional space according to a switch code (control code) contained in land object data explained before. This process is explained in detail by Figures 27 to 36 below.

In step S7, a camera process is carried out. For example, a coordinate of a visual point to the object is calculated such that a line or field of sight as viewed through a viewfinder of the virtual camera comes to an angle designated through the joystick 45 by the player.

In step S8, RSP 122 performs a rendering process. That is, RCP 12, under the control of CPU 11, performs transformation (coordinate transformation and

frame memory rendering) on the image data to display a movable object and stationary object based on the texture data for the movable object (e.g., an enemy object, player object, etc.) and the stationary object (e.g., background) stored in image data area 201 of RAM 14. Specifically, colors are given to polygons for each movable object and stationary object.

In step S9, CPU 11 performs a sound process based on sound data (e.g., messages, melody, sound effects, etc). In particular, BGM and the like are switched over according to a melody code (control code) previously set in the land object, as shown in a subroutine of Figure 7.

In step S10, CPU 11 reads out image data stored on frame memory area 203 of RAM 14, according to a result of the rendering process of the step S8. Thus, a player object, moving object, stationary object and enemy object, and the like are displayed on a display screen of display 30 in Figures 1 and 2.

In step S11, RCP 12 reads out the sound data obtained as a result of the sound processing of the step S9, thereby outputting sound (e.g., melody, sound effects, conversation, etc).

In step S12, whether or not the course was cleared is determined (course clear detection). If the course was not cleared, it is determined in step S13 whether the game is over. If the game is not over, the process returns to step S2 to repeat steps S2 through S13 until a game over condition is detected. If a game over condition is detected (i.e., the number of mistakes permitted for the player reaches a predetermined number of times or the life of player object is

consumed by a predetermined amount), a game over process is performed. A game over process can be a selection of game play continuation or backup data memorization.

If a condition of clearing the course (e.g., defeating an enemy, etc.) is detected in step S12, the course clear process is carried out and thereafter the process returns to step S1.

Figure 8 is a subroutine of the land object process shown in step S3 of Figure 7. In step S301, CPU 11 in Figure 2 reads out polygon data of a land object required at that time. This data is transferred from image data area 23 in Figure 5 of external ROM 21 to image data area 205 in Figure 6 of internal RAM 14. This polygon data contains a control code previously set, as was explained before. Accordingly, if step S301 is executed, the same control data is simultaneously read out. The read polygon data containing a control code (e.g., action code, camera switch code, etc.) is temporarily held in display list area 201 of internal RAM 14.

In step S302, texture data is read out that corresponds to the land object and transferred to image data area 205 of internal RAM 14. In step S303, camera data is read out of image data area 205 which corresponds to that land object. These texture data and camera data are stored on display list area 201, similar to the polygon data.

Then, in step S304, the land object is stored in display list area 201. It is determined in step S305 whether or not the process of steps S301 to S304 has



been executed on all the land objects. If the determination is "NO", the process is again executed from step S301. If all the land objects has been completed of the process, i.e. if "YES" is determined, the subroutine of Figure 8 is ended and the process returns to the main routine.

The action determination process in step S4 of Figure 7 is carried out according to a flowchart shown in Figure 9. That is, in step S401, CPU 11 in Figure 2 detects a state of the player object (i.e., whether or not the player object is in any action). If the player object is in a course of an action, "YES" is determined in step S402, and the process advances to step S403.

In step S403, CPU 11 refers to the register/flag area 209 of RAM 14 shown in Figure 6 and detects a control code or action code contained in the object data of a land object existing at the foot of the player object. The control code or action code, as was explained before, has been previously set in land object area 23b of external ROM 21 shown in Figure 5 and previously transferred to image data area 205. The land object data is read onto display list area 201 in every frame, and CPU 11 detects an action code in display list area 201.

Subsequently, CPU 11 in step S404 detects whether the player object is falling. That is, the player object is determined in action in step S402, and it is determined whether or not the action is "fall".

If the player object is falling, then CPU 11 in step S405 detects the height of the player object at that time from the land object. CPU 11 in step S406

determines that the player object lands when the height of the player object from the land object is a predetermined height (i.e., the height is sufficiently small).

Next, CPU 11, in step S407, causes the player object to begin a landing action.

In step S407, CPU 11 causes the player object to change in form based on landing-action animation data stored in player object data area 23a of external ROM 201 and controls RCP 12 to write color data to frame memory area 203. This animation data represents the skeletal movement of a player object. The player object is displayed by a combination of the animation data and the polygon data, in a manner similar to the land objects. Accordingly, if animation data is different, the player object changes in action, even with same polygon data. In step S407, the player object can be caused to make a landing action by reading out animation data for "landing action".

If it is determined in step S402 that the player object action state is not "in the course of an action", CPU 11, in step S408, detects a control code or action code for a land object existing nearby (in front or at the foot of) the player object from display list area 201, similar to step S403. In step S409, CPU 11 refers to the attribute data of the land object at the foot of the player object and determines whether the land object is a "hollow" or "hole". Alternatively, whether the land object at that time is a hollow or hole may be determined by a floor object located at zero degrees (parallel or horizontal) with respect to a moving direction of the player object. The floor object is formed with a downward step.

If the land object is a "hollow" or "hole", CPU 11, in step S410, executes a "hole action" subroutine shown in Figure 10. If "NO" is determined in step S409, step S411 determines whether or not the land object is labeled "wall surface" by the attribute code. However, as stated before, a wall surface object may be detected by a 90 degree angle with respect to the player object's advancing direction or the floor object. If the land object is a "wall surface," CPU 11, in step S412, executes a "wall surface action" subroutine shown in Figure 16. If "NO" is determined in step S411, step S413 determines whether the land object is a "door" by the attribute code or by an angle to the floor object. If the land object is a "door," CPU 11, in step S414, executes a "door action" subroutine in Figure 23. If "NO" is determined in step S413, step S415 determines whether or not the land object is a "ladder" by an attribute code or by an angle to the floor object. If the land object is a "ladder", CPU 11 in step S416 executes a "ladder action" subroutine.

Figure 10, in conjunction with Figures 11 to 15, explains a "hole action." In step S417 of Figure 10, an action code or control code for the land object at the foot of the player object in front of the hole is detected from display list area 201. More specifically, if the attribute data of a floor object constituting a "hole" includes one, two, or more bits of a control code and the control code is "0," the control code is set as default to "jump." In addition to "jump," the control codes of a floor object constituting a hole include: "bottomless pit," "scene switching," "not-fall," "step off," etc.

If the control code or action code detected in step S418 is not a "not-fall" code (i.e., where the control code or action code is "jump"), "NO" is determined in step S418. In step S419, CPU 11 determines a height of the player object from a land object, in a similar manner to the previous step S405.

It is determined in step S420 whether or not the calculated height of the player object is lower than a predetermined height (e.g., "200 cm"). Note that "cm" is a virtual length unit within a virtual three dimensional space. If "NO" is determined in step S420, CPU 11 calculates, in step S421, the moving speed of the player object at that time. In step S422, CPU 11 calculates the distance the player object should jump based on the height calculated in step S419 and the speed calculated in step S421. In step S423, the action of a jump is started according to the jump distance.

Figure 11 shows an example of a jump action, in which the player object can jump across a hole to an opposite bank because of a short distance L1 of the hole. Figure 12 shows an example of a jump action, in which the player object cannot jump across the hole but can lay his hand on the opposite bank because the hole is an intermediate distance L2. Figure 13 shows an example of a jump in which the hole distance L3 is too long for the player object to jump across the hole or to lay his hand on the opposite bank, and the player object falls into the hole. In any of the cases, the jump action required is automatically effected according to a jump code contained in a land object existing thereon.

The distance that the player object can jump across is correlated to the moving speed of the player object. For example, if the player object is running fast, it can jump across a large hole. However, when the player object is walking slowly, the player object may not jump across the hole, even if the control code "jump" has been set. Consequently, when the player object is walking, the player object may not jump across but rather falls into the hole or hangs on the opposite cliff.

Such jump actions can be achieved by reading corresponding animation data from player object data area 23a of external ROM 21, as was explained before.

If "YES" is determined in step S418 (i.e., if the control code or action code of a land object in front of the hole is a "not-fall" code), CPU 11 in step S424 causes the player object to begin an action of not-fall. In this case, the player object is going to fall into the hole but instead hangs on the opposite cliff.

If, in step S420, the height of the player object is determined less than 200 cm, no jump should be effected. In step S425, CPU 11 starts the player object to make a falling action. Conversely, if the height or depth of the hole is greater than 200 cm (virtual length), a jump action as mentioned above is executed. If less than 200 cm, the player object is caused to walk into the hole without jumping (as shown in Figure 14).

If "NO" is determined in step S409, whether a kind of a land object is a "wall surface" is determined by referring to attribute data or an angle in step

S411. If "YES" is determined in step S411, CPU 11 in step S412 starts a "wall surface action" when the player object faces a wall surface. This wall surface action is executed according to a flowchart shown in Figure 15.

In step S426 of Figure 15, CPU 11 determines whether or not a control code or action code contained in a land object "wall surface" existing near the player object is "forbid" (i.e., the player object is forbidden from getting over a wall surface). If a "forbid" code is present, the process returns to the main routine.

If a control code or action code contained in each polygon constituting the wall surface is "climb", CPU 11, in step S428, causes the player object to perform a wall-surface climbing action, as shown in Figure 16. In the Figure 16 example, if the player object contacts a wall and is put on the wall surface, it moves over the wall surface in response to the player's joystick 45 operation. Turning joystick 45 upward causes the player object to climb up the wall surface, while turning joystick 45 downward causes the player object to move down. If the player object moves up to a wall surface position where the control code "climb" is not set, the player object can no longer climb on the wall surface, which results in falling down. If the wall surface object faced with the player object is set with an action code "climb", the player object automatically climbs up the wall surface. Nevertheless, the movement of the player object can be determined through joystick 45.

If the control code or action code of the wall surface object is not "forbid" or "climb" and a floor object in front of a wall surface object is set with control code "jump", CPU 11, in step S429, calculates a wall surface height. The player object automatically performs its optimal action in accordance with the calculated wall surface height, as hereinafter described.

In step S430, CPU 11 determines whether the calculated wall surface height lies within a range of 0 to 25 cm (i.e.,  $0 < H \leq 25$ ). A height in this range indicates a very low wall surface. In this case, the player object can get over the wall surface as if it were climbing up stairs. In step S431, CPU 11 reads required animation data out of external ROM 21, or RAM 14, causing the player object to begin a "going up stairs" action shown in Figure 17. In the Figure 17 example, the wall surface is small in height. Accordingly, the player object can get over the stairs as a wall surface by an action of climbing the stairs step-by-step according to the control code "jump" set in the floor object. In this case, the control code "jump" has previously been set in the floor object in front of the wall surface object, or stairs, as shown in Figure 17.

In step S432, CPU 11 determines whether or not the wall surface height is in a range of 25 cm to 50 cm (i.e.,  $25 < H \leq 50$ ). This range of height indicates a low wall surface, and the player object can get over the wall surface by jumping. Accordingly, CPU 11, in step S433, reads required animation data out of ROM 21, or RAM 14, causing the player object to begin a "jump" action shown in Figure 18. In Figure 18, the player object jumps at the front of the

wall surface, thus getting over the wall surface. In this case, a control code "jump" has previously been set in a land object, or floor object, in front of the wall surface object, as shown in Figure 18.

In step S434, CPU 11 determines whether or not the wall surface height is in a range of 50 cm to 100 cm (i.e.,  $50 < H \leq 100$ ). This range of height indicates a comparatively high wall surface, and the player object can get over the wall surface by light climbing. Accordingly, in step S435, CPU 11 reads out required animation data to cause the player object to begin a "light climb" action shown in Figure 19. In Figure 19, the player object puts his hands on the wall surface object so that the body is pushed up atop the wall surface through a hand's pulling force and a foot's jumping force. In this case, a control code "jump" has previously been set in a floor on this side of the wall surface, as shown in Figure 19.

In step S436, CPU 11 determines whether or not the wall surface height is in a range of 100 cm to 150 cm (i.e.,  $100 < H \leq 150$ ). This range of height indicates a high wall surface, and the player object can get over the wall surface by usual climbing. Accordingly, CPU 11, in step S437, reads out required animation data, causing the player object to begin a "middle climb" action shown in Figure 20. In Figure 20, the player object responds to a "jump" code contained in a floor object in front of the floor, and lightly jumps at the front of the objective wall surface put his hand on the top end of the wall surface. The



player object's feet cannot touch the ground so that the body is lifted to the wall top end only through the hand's chinning force.

In step S438, CPU determines whether or not the wall surface height is in a range of 150 cm to 250 cm. (i.e.  $150 < H \leq 250$ ). This range of height indicates an extremely high wall surface, and the player object can get over the wall surface by hard climbing. Accordingly, CPU 11, in step S439, causes the player object to begin an action "hard climb" shown in Figure 21. In Figure 21, the player object responds to a control code "jump" in a floor object in front of the objective wall surface, and makes a high jump to put its hand on a wall top end. The player object's feet are floating so that the body is lifted to the top of the wall through only a hand's pulling force.

In this manner, CPU 11 detects a control code or action code contained in the object data of a land object in the vicinity of the player object exists, whereby the player object is caused to make an action in accordance with the control code or action code, such as wall climbing in the embodiment. If the control code or action code contained in the wall surface object is "climb," getting over the wall surface is by "climbing" instead of "jumping," as was explained before. However, if a "forbid" code is embedded in the wall surface object, the player object is not allowed to get over the wall surface.

Figure 22 shows a subroutine for "door action" shown in step S414 of Figure 9. In step S440 of Figure 22, CPU 11 refers to controller data area 207 of RAM 14 and determines whether or not the player is operating A button 47A

shown in Figure 1. If A button 47A is not being operated, the player does not intend the player object to enter the door, and the process returns to the main routine.

If "YES" is determined in step S440, CPU 11 in step S441 refers to image data area 205 shown in Figure 6 and detects the door position from the coordinate position of a polygon constituting the door. In step S442, the player object is correctly positioned to open the door, based on the door position as detected in step S441. The corrected player object positioned is recorded in image data area 205. Thereafter, in step S443, a door action is carried out. The CPU 11 reads required polygon data or animation data from image data area 205, to make the player object perform a series of door actions (i.e., to grip a knob, open the door, enter the door, and close the door). Figure 23 illustrates the player object entering a door. In the "door action" to be executed in the step S413 of Figure 9, the player object automatically opens and closes a door according to a "door code" previously set in a land object shown in Figure 23.

The "door" is set as a control code or action code in a floor object immediately in front of the door in the above embodiment explanation. Alternatively, the "door" code may be set on the door object.

Figure 24 shows a detail of an action "ladder" to be executed in step S416 of Figure 9. In step S444 of Figure 24, image data area 205 shown in Figure 6 is referred to so that a ladder position is determined from the coordinate position of a polygon constituting the ladder. In step S445, the player object's position is

corrected such that player object is positioned at the foot of the ladder based on a ladder position detected by step S444. The corrected player object position is recorded in image data area 205. Thereafter, in step S446, a ladder action is carried out. The CPU 11 reads required polygon data or animation data from image data 205 and causes the player object to perform a series of ladder-climbing actions (i.e., to put hands and feet on the ladder and alternately move left and right hands and feet on the ladder). Figure 25 illustrates the player object climbing midway up the ladder. For the "ladder action" to be executed in step S416 of Figure 9, the player object automatically climbs the ladder according to a "ladder" code previously set in a land object shown in Figure 25, (i.e., a wall surface object).

Explaining in greater detail, in the Figure 25 embodiment no control code "step off" has been set in the floor object in front of the wall surface object constituting the ladder. Accordingly, the player object can make a "ladder climb" action according to a control code "ladder" for the wall surface object, and the floor object does not affect the ladder climb action unless the control code in the floor object is on "step off." If a "ladder" code is set on the wall surface, the player object automatically grips the ladder. Thereafter, when facing the wall surface the player object goes up the ladder if the player tilts joystick 45, shown in Figure 1, up and comes down if joystick 45 is tilted down. If the player object reaches the top of the ladder, the player object automatically steps on a floor close thereto. Conversely, if the player object comes from the

upper floor to the ladder, an "enter ladder" code set in a wall surface object behind the ladder is detected so that the player object can go down the ladder according to the code.

According to this embodiment, it is possible to cause the player object to automatically make a different action depending on a control code or action code contained in a land object where the player object exists. Accordingly, it is easy to control the action of the player object with a program setting.

A flowchart shown in Figure 26 represents a player object processing operation for step S5 of the main routine of Figure 7. In step S501, CPU 11 determines whether or not the player object is in a course of action. If in a course of action, the position and pose of the player object are determined so that the player object continues its action. The pose is determined by animation data explained above.

If the player object is not in a course of action, CPU 11 in step S503 detects an operation state of joystick 45 shown in Figure 1, and Figure 4 included in controller 40. Subsequently, the moving direction, moving speed and position and pose of the player object are determined respectively in steps S503, S504 and S505, according to an operation state of joystick 45. In step S507, the player object is registered to the display list area 201 shown in Figure 6 of RAM 14, similar to the case after passing through step S502. Accordingly, the player object is displayed depending on the operation state of joystick 45.

The camera determination process in step S6 of Figure 7 is explained in detail with reference to Figure 27 as well as the related figures. In step S601 of Figure 27, CPU 11 makes reference to the data in image data area 205, and detects a control code (i.e., a camera code) previously set in the object data of a land object existing underneath the player object. In each of steps S602, S604, S606, S608 and S610, it is determined whether the detected control code is a first camera code, second camera code, third camera code, fourth camera code or fifth camera code.

Explanation is made herein on a first camera, second camera, third camera, fourth camera, and fifth camera which have been placed in the virtual three dimensional space in the embodiment, based on Figure 28. In the example of Figure 28, a longitudinal wall is provided in almost a center of a space that is rectangular in plan, wherein a door is formed on one part of the wall. A third camera is fixed and directed at the side of the door opening. On the opposite side of the door, a fourth camera is set up. This fourth camera is a zoom camera to view a player object opening and entering the door. Also, a second camera and a fifth camera are individually fixed in two respective corners in the space. The first camera is a movable camera which follows the player object. Camera control is explained below on an assumption of this embodiment having the five virtual cameras in the three dimensional space as above. However, the number, arrangement and function or roll (i.e., fixing, moving, zooming, etc.) can be

appropriately modified as required, and the embodiment is not intended to be limited to the described arrangement.

Note that in Figure 28 the terms "first camera", "second camera", . . . , "fifth camera" given in blocks (rectangular lattices) represent respective control codes, or camera codes, previously set in the land objects of this three dimensional space. Consequently, if the player object exists in one block, the player object will be viewed by a camera corresponding to the camera code contained in that block.

Referring back to Figure 27, if a first camera code is detected in step S602, in step S603, a first camera control program is selected. The camera control program, as explained before, is set in camera control program area 22f shown in Figure 5 of external ROM 21, which is transferred as required to program area 202 of internal RAM 14. Accordingly, CPU 11 in step S603 reads a first camera control program out of program area 202 shown in Figure 6.

The first camera control program causes the first camera to follow the player object as described before. In the first camera control program detailed in Figure 29, in step S612, the data in the image data area 205 shown in Figure 6 is referred to and the position of the player object is detected. Figure 30 illustrates a player object from the perspective of a first camera according to the Figure 29 flowchart. In step S613, CPU 11 determines the position of the first camera such that the distance from the player object shown in Figure 30 to the first camera remains constant. In step S614, the first camera faces the direction

of the player object. Accordingly, the first camera takes a player object-back view with a constant distance.

In a second camera control program to be executed in step S605 in Figure 27, in step S615 a position of the player object is detected as shown in Figure 31, similar to step S612 in Figure 29. Then, in step S616, the second camera faces the direction of the player object. That is, the second camera views the player object from a fixed position as shown in Figure 28.

Since the fifth camera is a fixed camera like the second camera, a fifth camera control program selected in step S611 is similar to the second camera control program of Figure 31.

The third camera is fixedly set up in front of the door as shown in Figure 28. Accordingly, the third camera merely takes the player object entering or exiting the door from a fixed distance point. The third camera control program of step S607 in Figure 27 includes step S617 in Figure 32. In step S617, the third camera faces the door. Accordingly, the player object entering or exiting the door is taken by the third camera, as shown in Figure 33.

Figure 34 shows the detail of the fourth camera control program executed in step S609 of Figure 27. As will be well understood from Figure 28, the fourth camera is chosen when the player object has entered a block that has a fourth camera code set in it. In step S618 of Figure 34, the number of frames is detected after detecting a fourth camera code and step S609 is entered (i.e., after camera change over), because there are two ways in which the fourth camera is

selected to view the player object. If the number of the frames is less than a predetermined number (i.e., immediately after a camera change over), "YES" is determined in step S619. In this case, CPU 11, in step S620, controls the fourth camera such that the fourth camera views, from a predetermined position, the player object entering the door. The player object taken by the fourth camera in step S620 is illustrated in Figure 35. As will be understood from Figure 35, the fourth camera is fixed at the position shown in Figure 28, and, in step S620 immediately after the camera change over, the fourth camera takes a distant view of the player object entering the door. The fourth camera views a comparatively wide range including the player object. Consequently, if the player object is entering the door as described in this embodiment, the player can readily understand where the player object, or hero, exists from an overall view.

Before a predetermined number of frames or time elapsed from the camera changed over and, but not immediately after that camera changed over, "NO" is determined in step S621. In step S622, CPU 11 causes the fourth camera to zoom so as to take a close-range view of the player object, as shown in Figure 36. The view is in a comparatively narrow range including the player object.

If a predetermined number of frames has elapsed, "YES" is determined in step S621. In this case, CPU 11 switches from the fourth camera to the first camera, as shown in step S623.



According to this embodiment, it is possible to automatically switch the camera over to take the player object and its function depending on a control code, or camera code, contained in a land object where the player object exists. Consequently, even where complex camera switching is necessary, it is very easy to set up a program therefor. The camera is switched depending on the X-Y coordinate position of the player object, and camera switching is the same in the same X-Y coordinate, irrespective of Z coordinate (i.e., height). In the method of this embodiment, the camera switching codes are embedded in the land objects. Accordingly, in the case of a same X-Y plane but different height (Z), it is possible to set a different land object (i.e., camera code) and hence a different camera. That is, in the embodiment, camera switching is feasible in a three dimensional fashion.

Incidentally, after ending any of steps S620, S622 or S623, the process returns to the main routine.

Referring to Figure 37, the explanation is made in detail on step S9 of the Figure 7 main routine (i.e., sound processing). This sound processing utilizes the control codes explained before. Consequently, in step S901 of Figure 37, reference is made to image data area 205 to detect a control code, or melody code, set on a land object where the player object exists. In step S902, it is determined whether or not the control code, or melody code, is on BGM1. The melody code BGM1 is a code to select the first BGM (Back-Ground Melody). Consequently, after "YES" is determined in step S902, CPU 11 in step S903

reads melody or sound data for the first BGM out of sound memory area 206 shown in Figure 6 and outputs it to bus control circuit 121 in Figure 2.

If "NO" is determined in step S902, it is determined in step S904 whether or not the control code, or melody code, is for BGM2. The melody code BGM2 is a code to select the second BGM. Accordingly, after "YES" is determined in step S904, CPU 11 in step S905 reads melody or sound data for the second BGM out of sound memory area 206, and outputs it to bus control circuit 121.

After "NO" is determined in both steps S902 and S904, CPU 11 in step S906 determines whether or not the control code, or melody code, is for "outcry". The melody code "outcry" generates an outcry sound effect. Consequently, after "YES" is determined in step S906, CPU 11 in step S907 reads sound data for "outcry" out of sound memory area 206, and outputs it to bus control circuit 121.

If the control code, melody code or sound code is different from the ones described above, in step S908 another code of sound data is set.

In this manner, according to this embodiment, it is possible to automatically switch the sound in accordance with a control code, or sound code, contained in a land object where the player object exists. Accordingly, where switching sound requires difficult control, it is easy to setup a program therefor.

Although the present invention has been described and illustrated in detail, it is clearly understood that the same is by way of illustration and

example only and is not to be taken by way of limitation. The spirit and scope of the present invention is limited only by the terms of the appended claims.